USING ARTIFICIAL INTELLIGENCE FOR OBJECT DETECTION (SECOND PART)

A MESTERSÉGES INTELLIGENCIA FELHASZNÁLÁSI LEHETŐSÉGEI AZ OBJEKTUMFELISMERÉSBEN (MÁSODIK RÉSZ)

KOLLÁR Csaba[1] – NAGY Barna[2]

Abstract

In the first part of our study, we dealt with machine vision, the history of technical and information science of machine vision, neural networks, and the teaching of networks. The main topics of the second part of our paper are the support of machine vision with neural networks, the framework of neural networks, the presentation of the platform for practical development, the detection of objects in moving images, and finally the thoughts that conclude our two-part study. There are a relatively large number of hardware and software solutions to choose from for using artificial intelligence in object recognition. The strength of the solution we have chosen is that it provides a relatively inexpensive solution that can inspire not only practitioners but also students to try out the development and test environment presented and to think further about the possibilities.

Keywords

security systems, artificial intelligence, computer vision, Intel Neural Compute Stick, Raspberry Pi

Absztrakt

Tanulmányunk első részében a gépi látással, a gépi látás technika- és információtudományi történetével, a neurális hálózatokkal, a hálózatok tanításával foglalkoztunk. Írásunk második részének fontosabb témái a gépi látás támogatása neurális hálózatokkal, a neurális hálózatok keretrendszere, a gyakorlati fejlesztést biztosító platform bemutatása, az objektumok detektálása mozgóképen, s végül a kétrészes tanulmányunkat záró gondolatok. A mesterséges intelligencia objektumfelismerésben történő használatára viszonylag sok hardveres és szoftveres megoldás közül lehet választani. Az általunk választott megoldás erősségét az adja, hogy használatával egy relatíve olcsónak tekinthető megoldás áll rendelkezésre, amelyik nem csak a gyakorló szakembereket, hanem a hallgatókat is inspirálhatja a bemutatott fejlesztési- illetve tesztkörnyezet kipróbálására és a benne levő lehetőségek továbbgondolására.

Kulcsszavak

biztonságtechnika, mesterséges intelligencia, gépi látás, Intel Neural Compute Stick, Raspberry Pi

[1] kollar.csaba@uni-obuda.hu | ORCID: 0000-0002-0981-2385 | senior research fellow, Óbuda University Bánki Donát Faculty of Mechanical and Safety Engineering | tudományos főmunkatárs, Óbudai Egyetem Bánki Donát Gépész és Biztonságtechnikai Mérnöki Kar

[2] nagy.barna@blx.hu | ORCID: 0000-0002-8101-1080 | software architect, Blumenthal Consulting Kft. | szoftver architect, Blumenthal Consulting Kft.

A GÉPI LÁTÁS TÁMOGATÁSA NEURÁLIS HÁLÓZATOKKAL

Az utóbbi években a Computer Vision alkalmazásokban uralkodó lett a konvolúciós neurális hálózatok (Convolutional Neural Networks (CNN)) használata. A CNN egy deep neural network (DNN), azaz mély neurális hálózat. A hálózatokat többnyire felügyelt tanítással teszik alkalmassá egy-egy konkrét feladat elvégzésére.

A neurális hálózatok tanítása meglehetősen erőforrásigényes feladat. Általában nagy mennyiségű képre (tanító pontokra) és komoly számítási kapacitásra van szükség az elvégzéséhez. Ugyanakkor a hálózat használata, az előhívás (inference) sokkal kevesebb erőforrást igényel. Már edge eszközökben is futtatható és a hálózat válasza (alkalmazástól függően) akár milliszekundum alatt előáll. Mind a tanításra, mind az előhívásra (inference) igénybe szoktak venni hardveres gyorsítókat (accelerator), melyek a CNN számításait segítik. [1]

A hálózatok tanítása sohasem egy lépésben, hanem iteratívan, lépésenként történik, amely több fogalmat is bevezetett. Egy epoch a teljes tanítóminta-készlet (training dataset) egy alkalommal történő „végigtanítását" jelenti. Egy tanítóminta (bemenet és kimenet páros) tanítása ebben az esetben egy előreterjesztés (forward propagation) és egy hiba-visszaterjesztés (backward propagation of errors) folyamatot jelent.

Az előreterjesztés során a hálózat bemenetére illesztett bemeneti érték alapján számoljuk a hálózat aktuális válaszát. A visszaterjesztés során pedig a számított aktuális válasz és a tanítómintában lévő elvárt válasz közötti eltérés alapján kismértékben módosítjuk a többrétegű hálózat kapcsolati súlyait.

A neurális hálózatok néhány keretrendszere

A neurális hálózatok elterjedése magával hozta az azokat támogató, modellező rendszerek nagy számát is. Részben üzleti, részben más megfontolásból, de sokan fogtak ilyen rendszerek fejlesztésébe. A későbbi vizsgálatokhoz célszerű kiválasztani egyet, mely illeszkedik az elvárásainkhoz és a lehetőségeinkhez. Összegyűjtöttünk néhány ismert rendszert és kiértékeltük őket több szempont alapján. Az általunk vizsgált keretrendszerek a következők voltak:

Google Tensorflow. A Google fejlesztőcsapata által készített nyílt forráskódú TensorFlow az egyik legnépszerűbb machine learning platform. Az eszköz a modellezést, a tanítást és az éles környezetben történő futtatást is támogatja. A TensorFlow nemcsak neurális hálózatok, azon belül is deep neural network modellezésére használható. Egy általános eszköz, mellyel számos machine learning feladat megoldható, ráadásul csekély változtatással sokféle eszközön is futtatható. [2]

Apache MXNet. Az Apache alapítvány egyik projektje, melyben egy skálázható mélytanuló rendszert fejleszthettek. Előnye, hogy nagyon sok nyelven elérhető hozzá interfész, továbbá nagyon könnyen futtatható egyszerre több eszközön is. [3]

ONNX. Az Open Neural Network Exchange (ONNX) egy nyílt formátum, melyet eredetileg a Facebook és a Microsoft készített, hogy a már meglévő gépi tanulást segítő keretrendszerek közötti átjárást segítsék. A formátum eszközei támogatják a legtöbb keretrendszert, például Caffe, Keras, MXNet, PyTorch, TensorFlow és továbbiakat. [4]

Keras. A Keras egy Python nyelven írt, nyílt forrású, neurális hálózat könyvtár (library), amely más keretrendszerekhez (pl. TensorFlow) biztosít magasszintű hozzáférést. A TensorFlow a saját high-level API-jának választotta, de más rendszerekkel is integrálható. [5]

Caffe. A Caffe (Convolutional Architecture for Fast Feature Embedding)) keretrendszert eredetileg Yangqing Jia a Berkeley egyetem PhD hallgatója készítette konvolúciós neurális hálózatok modellezésére és futtatására. Előnyei közzé tartozik, hogy nagyon könnyű vele modelleket készíteni, ami gyakorlatilag egy szöveges állomány. Meglehetősen elterjedt akadémiai és egyéb kutató projektekben, ezért rengeteg modell érhető el a formátumában, illetve nagyon sok más modellt implementáltak újra a Caffe rendszerében. [6]

Darknet. A Darknet egy nyílt forráskódú neurális hálózati keretrendszer, amelyet C-ben és CUDA-ban írtak. Gyors, könnyen telepíthető, és támogatja a CPU és a GPU számítását. Legfontosabb tulajdonsága, hogy osztályozza a képeket olyan népszerű modellekkel, mint a ResNet és a ResNeXt. [7]

A felsorolt rendszerek közül kiemelkedik a TensorFlow, mely sokoldalúsága és dokumentáltsága a többi rendszer elé helyezi. Ugyanakkor komplexitása nagy, a betanulási, megismerési idő sokáig tart. Mindezek mellett – ahogy írtuk – a Caffe is nagy mértékben elterjedt a tudományos és startup körökben. Nem olyan univerzális, mint a TensorFlow, viszont pont arra alkalmas, amire mi szeretnénk használni (konvolúciós neurális hálózatok). Használata egyszerű, számos cikk, publikáció foglalkozik vele. A számunkra fontos OpenCV és OpenVINO támogatással is rendelkezik, így ezt a keretrendszert választottuk, annak ellenére, hogy jelenleg nem fejlesztik. A vizsgálatokban használt hálózatokat Caffe formátumban kerestük, amiket közvetlenül és (szükség esetén) konvertálva használtunk. Az első táblázatban a fontosabb keretrendszerek összehasonlító elemzését ismertetjük.

Elnevezés	**TensorFlow**	**MXNet**	**ONNX**	**Keras**	**Caffe**	**Darknet**
Aktuális verzió	2.4.1	1.8.0	1.9.0	2.4.0	1.0 (2.0)	
Utolsó verzió	2021.01.21.	2021.03.03.	2021.04.19.	2020.06.17.	2017.04.18.	kb. 2 éve
Fejlesztő	Google BrainTeam	Apache Foundation	Facebook, Microsoft	François Chollet	Berkeley Vision and Learning Center	Joseph Redmon
Megjelenés	2015.11.09.	2017.09.05.	2017 szept.	2015.03.27.	2017.04.18.	2016
API nyelve	Python, C/C++ (C++, Go, Java, Javascript, Swift, stb.)	Python, C++, Java, Scala, R, Perl	Python, C++	Python	Python, C++	C++

Elnevezés	TensorFlow	MXNet	ONNX	Keras	Caffe	Darknet
Operációs rendszerek	Linux, macOS, Windows, Raspbian[3]	Linux, macOS, Windows, Raspbian	Linux, Windows	Linux, macOS, Windows	Linux, macOS, Windows	Linux, macOS, Windows
Futtató eszközök	CPU, Nvidia CUDA based, GPU, TPU[4]	CPU, GPU, Nvidia Jetson		CPU, GPU, TPU	CPU, GPU	GPU, (CPU)
OpenCV támogatás	igen[5]	nincs	igen	nincs	igen	igen
OpenVINO támogatás	igen, az OpenVINO közvetlenül konvertál TensorFlow modelleket	igen, az OpenVINO közvetlenül konvertál MXNet modelleket	igen, az OpenVINO közvetlenül konvertál ONNX modelleket	nincs közvetlen konverzió	igen, van közvetlen konverzió	nincs közvetlen konverzió
Nyílt forráskód	Apache License 2.0	Apache License 2.0	MIT License	igen	BSD License	igen
Elérhető modellek	nagyon sok	sok	közepes	sok	nagyon sok	pár darab

1. Táblázat: A főbb neural networks keretrendszerek paraméterei (saját összehasonlító elemzés és szerkesztés)

AZ INTEL NEURAL COMPUTE STICK 2 VIZSGÁLATA

Funkcionális tesztelések, az OpenVINO toolkit vizsgálata

Az alábbi alkalmazások, illetve modellek futtatásával azt vizsgáltuk, hogy az OpenVINO Toolkit fejlesztőeszköz egyes eszközei megfelelően működnek-e a környezetemben.

A hello_query_device alkalmazás. Az alkalmazás kilistázza az OpenVINO által elérhető eszközöket. Ezeken a hardvereken tudunk az Inference Engine segítségével modelleket futtatni.

Az alkalmazás futtatása:

```
python3 hello_query_device.py
```

Az alkalmazás segítségével több környezetben is lekérdeztük az elérhető eszközöket:

Environment / Computer (környezet/számítógép)	Available device (lehetséges eszköz)	Full device name (az eszköz teljes neve)
Mac mini (Mid-2011)	CPU	Intel(R) Core(TM) i5-2415M CPU @ 2.30GHz
Mac mini (Mid-2011)	MYRIAD	Intel Movidius Myriad X VPU
Raspberry Pi 3B	MYRIAD	Intel Movidius Myriad X VPU
Macbook Pro (Mid-2014)	CPU	Intel(R) Core(TM) i5-4278U CPU @ 2.60GHz

[3] A Raspberry Pi-re fordított Debian alapú Linux operációs rendszer

[4] TPU – Tensor Processing Unit: a Google által fejlesztett alkalmazásspecifikus IC, amit elsősorban számítóközpontokba szánt, de létezik belőle *edge* változat is (https://cloud.google.com/edge-tpu/).

[5] az OpenCV be tudja tölteni a modellt a *cv2.dnn.readNetFromTensorflow()* függvényhívással

Macbook Pro (Mid-2014)	MYRIAD	Intel Movidius Myriad X VPU

2. Táblázat: A hello_query_device alkalmazás segítségével történő lekérdezés (saját szerkesztés)

OpenVINO Model Downloader. A Model Downloader parancssori eszközzel számos open source, előre tanított (pre trained) hálózat modelljét lehet letölteni. Ezek a modellek valamelyik népszerű modellező eszköz (pl. Caffe, TensorFlow) formátumában vannak, ezért konvertálni kell azokat az Intermediate Representation (IR) formátumra. Ez az a formátum, amit az OpenVINO csomagban található Inference Engine (IE) fel tud olvasni, illetve futtatni tud a támogatott eszközön.

Az eszköz az alábbi útvonalon érhető el:

```
$INTEL_OPENVINO_DIR/deployment_tools/open_model_zoo/tools/downloader/downloader.py
```

Az AlexNET modell letöltése (half-precision FP16 / Caffe Model):

```
./downloader.py --name alexnet --output_dir . --precisions FP16

################|| Downloading models ||################

========== Downloading my_dir/public/alexnet/alexnet.prototxt
... 100%, 3 KB, 12302 KB/s, 0 seconds passed

========== Downloading my_dir/public/alexnet/alexnet.caffemodel
... 100%, 238146 KB, 2930 KB/s, 81 seconds passed
```

Az AlexNet modell tesztelése

Az AlexNet egy régi és ismert neurális hálózat. Már több, sokkal jobban teljesítő társa is elérhető, de a népszerűsége és a fórumokban fellelhető gazdag dokumentáltsága ideálissá teszi arra, hogy ennek segítségével vizsgáljuk az Intel Neural Compute Stick performanciáját. Az AlexNet az a konvolúciós neurális hálózat (CNN), mely 2012-ben megnyerte az ImageNet (ImageNet Large Scale Visual Recognition Challenge – ILSVRC) versenysorozatot. A modellt az OpenVINO Model Zoo-jából töltöttük le Caffe formátumban, ezért konvertálnunk kellett IR-re (Intermediate Representation):

```
python3 mo_caffe.py \
--input_model ./alexnet/alexnet.caffemodel --output_dir . \
--data_type=FP16 \
--model_name=alexnet_fp16

./mo.py \
--input_model ../tools/model_downloader/public/alexnet/alexnet.caffemodel \
--input_proto ../tools/model_downloader/public/alexnet/alexnet.prototxt \
--model_name=alexnet_fp32
```

A modell image classification-re használható, ezért egy alkalmas kép segítségével teszteltük a funkcionális működését:

1. Ábra: Dalmata fajtájú kutya (forrás: Hungarovet Állatkorház honlapja[6])

A hálózat egy 1000 elemű vektort ad vissza, az egyes címkék[7] valószínűségével. A tesztelés során egy Intel CPU-n és a Movidius VPU-n is futtattuk a hálózatot:

index	valószínűség	címke	index	valószínűség	címke
251	0,9404	dalmatian, coach dog, carriage dog	251	0,9390	dalmatian, coach dog, carriage dog
246	0,0498	Great Dane	246	0,0509	Great Dane
176	0,0053	Saluki, gazelle hound	176	0,0052	Saluki, gazelle hound
MYRIAD (Movidius VPU)			CPU (Mac mini 2011)		

3. Táblázat: Az AlexNet funkcionális tesztelése két eszközön (saját szerkesztés)

Mindkét eszközön nagyságrendileg azonos „magabiztossággal" találta el a hálózat, hogy mi található a képen. A kép eredeti mérete 512 * 302 pixel, amit a teszt szkript a hálózat bemenetére vág: 227 * 227 képpontra. A hálózat adott eszközön, minden futtatásakor ugyanazokat az valószínűségeket adta vissza.

Az AlexNet performancia vizsgálata. Az OpenVINO saját eszközt ad a teljesítmény vizsgálatokhoz. Ez a benchmark_app, melyet Python és C++ nyelven elérhet a fejlesztő. Ez utóbbit, használat előtt szükséges lefordítani az adott környezetre, mert a C++ mintaprogramoknak csak a forráskódja található meg a csomagban.

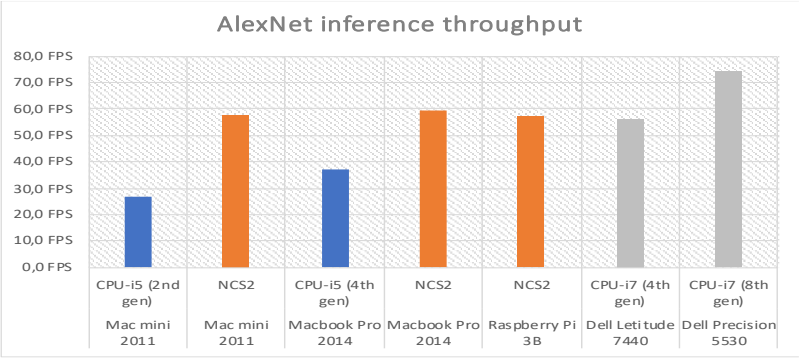
A teljesítménymérő alkalmazás futtatásához több paramétert is meg kell adni:

- *model:* a teszteléshez használt IR formátumú modell
- *images:* a teszteléshez használt kép vagy képek sorozata
- *api:* szinkron vagy aszinkron hívással érje el az Inference Engine-t
- *device:* az eszköz, amin futtatni kell a mérést

```
python3 ../tools/benchmark_tool/benchmark_app.py \
-m alexnet.xml \
-i ~/Downloads/alexnet/dog_03.jpg \
-d CPU -api async
```

[6] Webes elérhetőség: http://vetnetinfo.com/tudasbazis/2016/03/09/dalmata-dalmatian-kutya/
[7] Az ImageNet1000 classification indexei: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

A teljesítményt ötféle környezetben vizsgáltuk. Négy esetben személyi számítógépen (CPU és Neural Compute Stick 2) és az ötödik esetben pedig a Raspberry Pi kártyaszámítógépen. A benchmark_app 60 másodpercig futott aszinkron módban és ezalatt az idő alatt igyekezett minél többször feldolgozni ugyanazt a képet. A futtatások száma alapján volt kalkulálható az egységnyi időre jutó képfeldolgozások száma: FPS (frame per sec).



2. Ábra: Az egyes konfigurációk képfeldolgozási teljesítménye az AlexNet hálózaton mérve (saját szerkesztés)

A méréseket többször elvégeztük. Az eszközök teljesítménye (throughput in FPS) a számítógépek aktuális állapotától és egyéb tevékenységétől függően ingadozott. Az összesítésben egy átlagos értéket vettünk figyelembe.

A Neural Compute Stick 2 közel azonos teljesítményt nyújtott (narancs színű oszlopok) mindhárom számítógépben, és egyben számottevően jobb volt, mint a régebbi Intel CPU-k (kék oszlopok). A mért eredményekhez hozzátettünk két (az interneten publikált[8]) teszteredményt is. Ezek csak korlátozottan hasonlíthatóak össze a mi eredményeinkkel, mert nem ugyanabban, de nagyon hasonló tesztkörnyezetben készültek. Jól mutatják, hogy az újabb, erősebb Intel CPU-k nagyobb teljesítményre képesek, mint az NCS2.

A YOLOv3 modell tesztelése

A YOLO neurális hálózat. A YOLO (You Only Look Once) egy objektum detektáló neurális hálózat, illetve módszer, melyet Joseph Redmon publikált[9] negyedmagával együtt 2016-ban. A módszer újszerűsége (és a nevét is innen kapta), hogy az objektum detektálás összes lépését (localization + classification) a hálózat egy kiértékelése alatt teszi meg. A hálózat gyors elterjedése a kiemelkedő sebességének köszönhető. [8] Az Intel NCS2 további vizsgálatához előzetesen jó ötletnek gondoltuk, hogy egy népszerű és valós idejű objektum detektálást ígérő hálózattal teszteljük az eszközt.

[8] Jonas Werner (Cloud Solutions Architect) blogja: https://jonamiki.com/2019/02/27/trying-out-the-intel-neural-compute-stick-2-movidius-ncs2/
[9] Joseph Redmon YOLO weboldala: https://pjreddie.com/darknet/yolo/

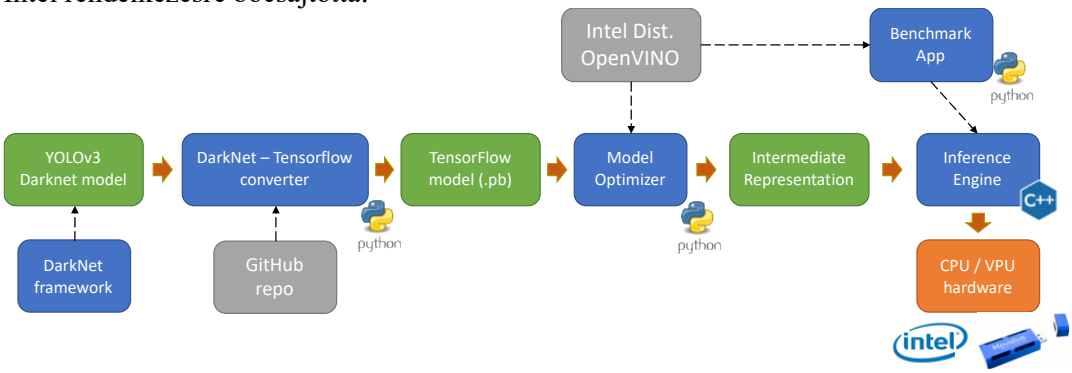
A YOLOv3 modell konvertálása IR formátumra. A YOLO v3-as verziója is a Darknet neurális hálózat modellező eszközben készült. Ez közvetlenül nem futtatható az OpenVINO Inference Engine-jén, ezért két lépésben konvertálni kellett az IR (Intermediate Representation) formátumra. A konverzióhoz a modell készítőjének weboldaláról[10] letöltöttük a betanított modellt (.weights fájl), valamint egy „külső[11]" eszközt, mellyel a Google TensorFlow modellezőjének a formátumára konvertáltuk. A TensorFlow model a gyakorlatban egy .pb (Protocol Buffer) kiterjesztésű fájlt jelentett.

```
# konvertáló eszköz letöltése
git clone https://github.com/mystic123/tensorflow-yolo-v3.git

# címkék letöltése
wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
# súlyok letöltése
wget https://pjreddie.com/media/files/yolov3.weights

# konvertálás TensorFlow modellre
python3 convert_weights_pb.py --class_names coco.names --data_format NHWC \
--weights_file yolov3.weights
```

Ez a modellek közötti konverzió rámutatott a különböző modellező keretrendszerek közötti átjárhatóság korlátoltságára. Például a Region réteg (layer) létezik a Darknet eszköztárában, de jellemzően nincs benne más keretrendszerekben, pl a TensorFlow-ban sem, így azokat egyedi implementációval szükséges pótolni. Ezt az egyedi implementációt az Intel rendelkezésre bocsájtotta.



3. Ábra: A Darknet YOLO modelljének konverziója és futtatása (saját szerkesztés)

A TensorFlow-IR konverzióra már az OpenVINO saját eszközét használtuk, a Model Optimizer-t.

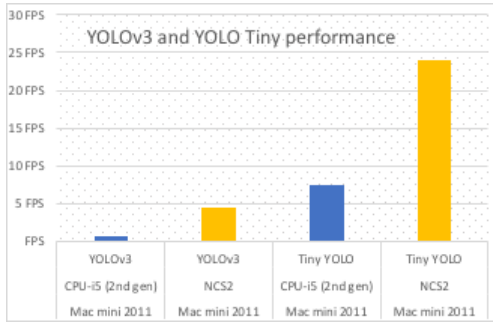
```
# konvertálás IR modellre (Model Optimizer)
python3 mo_tf.py \
--input_model ~/tensorflow-yolo-v3/frozen_darknet_yolov3_model.pb \
--tensorflow_use_custom_operations_config ~/tensorflow-yolo-v3/yolo_v3.json \
--batch 1
```

[10] A YOLO hálózat weboldala: https://pjreddie.com/darknet/yolo/
[11] Paweł Kapica (mystic123): GitHub oldal: https://github.com/mystic123

A YOLOv3 modell performancia vizsgálata. A YOLO modell performancia vizsgálatát az OpenVINO benchmark_app eszközével végeztük el, hasonlóan paraméterezve, mint ahogy az AlexNet esetében tettük.

```
python3 benchmark_app.py \
-m ~/Downloads/yolov3/frozen_darknet_yolov3_model.xml \
-i ~/Downloads/alexnet/dog_03.jpg \
-d CPU
```

A hálózat vizsgálat közben 60 másodperc alatt minél többször próbálta feldolgozni ugyanazt a képet. A vizsgálatot 2 környezetben végeztük el, egy személyi számítógép CPU-ját és az Intel NCS2-t tesztelve. Bár a hálózat a sebessége miatt lett igazán népszerű, a teszteredmények ezt nem igazolták. Az eredeti YOLO v3-as hálózat (a vizsgált környezetekben) nem alkalmas valós idejű feldolgozásra. A YOLO hálózatokhoz készített a fejlesztő egy-egy egyszerűsített változatot is. Ezek közül a YOLOv3 Tiny-at próbáltuk ki. Ez már sokkal jobb teljesítménnyel futott, ugyanakkor a hálózat „Tiny" verziója sokkal pontatlanabb, mint az eredeti változat.



4. Ábra: A YOLOv3 teljesítménye (OpenVINO benchmark_app) CPU-n és az Intel NCS2-n (saját szerkesztés)

Környezet	Modell	Iterációk száma	Performancia (FPS)
Mac mini 2011 CPU-i5 (2nd gen)	YOLOv3	44	0,62
Mac mini 2011 NCS2	YOLOv3	276	4,46
Mac mini 2011 CPU-i5 (2nd gen)	YOLOv3 Tiny	456	7,38
Mac mini 2011 NCS2	YOLOv3 Tiny	1452	24,07

4. Táblázat: Az egyes környezetekben mért teljesítmények (saját szerkesztés)

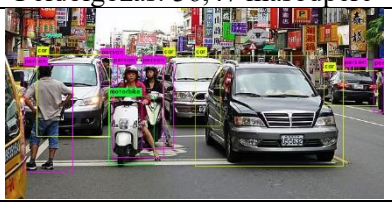
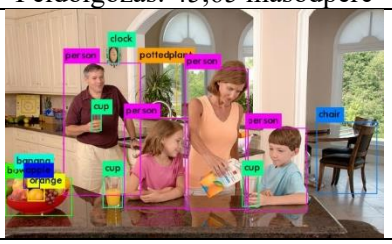
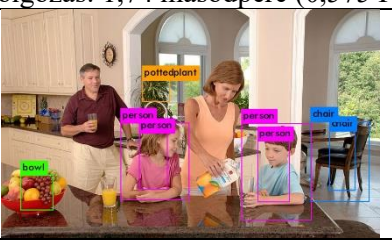
A mérési eredmények (ugyanabban a környezetben) minden alkalommal eltérőek voltak, ezért egy átlagos értéket tűntettük fel a táblázatban.

A YOLOv3 vizsgálata a Darknet környezetében. Feltételeztük, hogy a hálózat gyenge performanciáját a YOLOv3 modelljének dupla konvertálása okozhatta. Az eredeti hálózat (ahhoz, hogy használni tudjam) két lépésben is konvertálva lett, először a TensorFlow formátumára, majd az OpenVINO IR formátumra. Ahhoz, hogy ezt a feltételezésünk alá tudjam támasztani a Darknet keretrendszerében is kipróbáltuk a hálózatot. Első lépésben letöltöttük a rendszer forrását, majd az általunk használt asztali számítógépre fordítottuk:

```
git clone https://github.com/pjreddie/darknet.git
cd darknet
make
```

A modell ebben az esetben is nagyon lassan futott az asztali számítógép CPU-ján:

```
~/Downloads/alexnet/dog_03.jpg: Predicted in 36.684101 seconds.
dog: 100%
```

Darknet YOLOv3	Darknet YOLOv3 Tiny
	
Feldolgozás: 36,47 másodperc	Feldolgozás: 1,43 másodperc (0,70 FPS)
	
Feldolgozás: 45,65 másodperc	Feldolgozás: 1,74 másodperc (0,575 FPS)
	
Feldolgozás: 37,16 másodperc (0,027 FPS)	Feldolgozás: 1,51 másodperc (0,662 FPS)

5. Táblázat: A YOLOv3 és a YOLOv3 Tiny modellek pontosságának összehasonlítása Intel i5 CPU-n futtatva. (saját szerkesztés, képek forrása: www.pikrepo.com és Hungarovet Állatkorház)

A mérési eredmények alapján megállapítható, hogy a Darknet YOLOv3 hálózatának gyenge teljesítménye elsősorban nem a modellek konvertálásából fakad. A modell egyszerűen rosszul teljesít CPU-n futtatva, az Nvidia GPU-n viszont – a fejlesztő szerint – nagyságrendekkel gyorsabb [9] A YOLOv3 modellel kapcsolatos vizsgálódások több dologra mutattak rá. Az egyes modellek struktúrája, belső működése – implementációtól függően – speciális is lehet. Erre példa a Darknet rendszer Region layer-e, ami ebben a rendszerben jelent meg először. [10] Ez nemcsak a más modellező eszköz formátumára való konverziót nehezíti meg, de jelentős teljesítmény különbségek is felléphetnek, ha más eszközön is futtatni akarjuk. [9]

Objektum detektálása mozgóképen

Az eddigi vizsgálatok után egy mintaalkalmazás segítségével vizsgáltuk az Intel NCS2 eszközt. Az alkalmazás egy videostream-en végez objektum detektálási feladatot. Az objektum detektálást a MobileNet-SSD hálózattal végeztük. A hálózat a YOLO-hoz hasonlóan egy kép egyszeri futtatására (Single Shot) végzi el az objektum detektálást.

Az SSD hálózatok. Az SSD (Single Shot Detector) hálózatok a YOLO-hoz hasonlóan egy lépésben végzik el az objektumok lokalizációját és azonosítását (image classification). Első lépésben egy feature extraction történik a bemenetre helyezett képen, második lépésben pedig konvolúciós filterek segítségével add vissza téglalapokat (bounding boxes) és az azokhoz társított objektumokat (classes).

Mérés egy mintaalkalmazással. Az alkalmazás egy Python-ban írt szkript, mely megnyit egy videostream-et, minden egyes képkockára elvégzi az objektumdetektálást, majd ennek eredményét megjeleníti a képkockán. A megjelenítés a megtalált objektumok bekeretezését és címkézését jelenti. Az alkalmazást eredetileg úgy készítettük el, hogy egy webkamera képén dolgozott, de többszöri futtatás összehasonlíthatósága megkívánta, hogy mindig ugyanazon a videostream-en dolgozzon a szkript. A szkriptek szinkron, illetve aszinkron hívással érik el az IR-t (inference engine-t). A szinkron hívás során az OpenCV API-ját használtuk, az aszinkron hívásnál viszont az OpenVINO-ban lévő Python API-t. A méréshez használt videófájlt a Pikrepo[12] ingyenesen felhasználható videótárából töltöttük le.

A videó felbontása	320 x 240 pixel
Az osztályozás címkéinek száma	20 db (ennyi különböző objektumot tud felismerni)
Használt programozási környezet	Python (OpenCV és OpenVINO API-k)

6. Táblázat: A mérés paraméterei (saját szerkesztés)

A futtatás során arra voltunk kíváncsiak, hogy a környezetekben milyen sebességgel tudja a szkript feldolgozni a videó egyes képkockáit. A szemléletes eredményhez a mért feldolgozási időkből az egységnyi idő alatti feldolgozható képkockák (frame per sec) számát számoltuk.

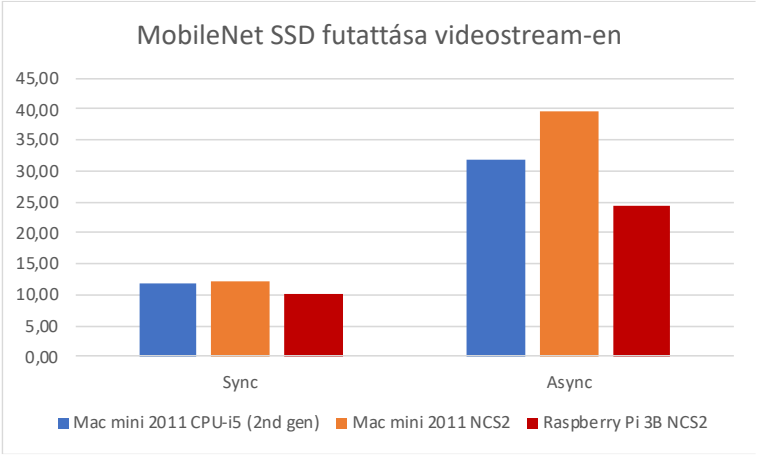
[12] https://www.pikrepo.com

5. Ábra: Képkocka feldolgozása videóból (saját szerkesztés)

	Szinkron feldolgozás sebessége (FPS)	Aszinkron feldolgozás sebessége (FPS)
Mac mini 2011 i5-CPU	11,92	31,70
Mac mini 2010 NCS2	12,24	39,52
Raspberry Pi 3B NCS2	10,21	24,51

7. Táblázat: Objektumdetektálás sebessége különböző környezetekben (saját forrás)

A mérési eredményeken szembeötlő, hogy az Inference Engine aszinkron elérése mennyivel gyorsabb feldolgozást eredményez. Az eddigi tapasztalatok alapján nem meglepő, hogy az Intel NCS2 eszköze jól teljesít a Intel második generációs i5-ös CPU-val szemben.



6. Figure: Objektumdetektálások összehasonlítása szinkron és aszinkron hívások kapcsán (saját forrás)

Az aszinkron hívásokkal megvalósított objektum detektálás már egy kis teljesítményű eszközön, a Raspberry Pi 3B-n is hasznosítható eredményt hozott. A 24 FPS feletti eredmény igazolja az Intel eszközének az – igaz, korlátozott – használhatóságát Edge eszközökben is. Az NCS2-t alkalmazó gépi látást használó alkalmazások tervezésekor figyelembe kell venni, hogy a mért eredmény egy kis felbontású videón készült, a felismerhető objektumok száma sem túl sok, ugyanakkor a Python interfészek használata (C++ helyett) is ront a teljesítményen.

Az alkalmazást (és teljesítményét) többféleképpen is tovább fejleszthetjük. A C++ API használatával elhagyhatjuk a Python okozta overhead-et, bár ez a C++ fordítási ideje miatt valamennyire lassabb fejlesztést teszt lehetővé. Egyszerre több Intel NCS2 használatával, az egymást követő képeket párhuzamosan tudjuk feldolgozni. Az NCS2 a számítógép USB interfészén csatlakozik, így egy alkalmas USB Hub segítségével könnyen több eszközt is csatlakoztatni tudunk a rendszerünkhöz. Az Intel által adott Inference Engine API támogatja több eszköz használatát, így szoftver oldalon is támogatott a „multi stick" felhasználás.

Az alkalmazás teljesítményére természetesen nemcsak a gyorsítóhardver és az azt meghívó kód van hatással, hanem a használt neurális hálózat is. Ahhoz, hogy az alkalmazásunkkal egy optimális teljesítményt érjünk el, a megfelelő neurális hálózat kiválasztása is fontos. Ez nemcsak a megfelelő struktúrát, hanem a megfelelő tanítást is jelenti. Azt láthattuk, hogy az egyszerűsített (pl.: YOLO Tiny) hálózatok hiába gyorsabbak, egyúttal pontatlanabbak is. Ezek a pontatlanságok részben javíthatók tanítással. Egy speciális feladatra szánt hálózatnak nem biztos, hogy több tucat vagy több száz objektumot kell felismernie, ezért érdemes a hálózatot az alkalmazásnak megfelelő módon tanítani. A tanítás azonban nagyon erőforrásigényes. Ezalatt nemcsak gépidő értendő, hanem a szükséges adatok (jelen esetben főleg képek) összeszedése, majd a tanítópontok összeállítása, amire sok-sok emberórát kell fordítanunk. A tanulmány készítése során csak előre tanított hálózatokat használtunk. Ez korlátozta a felhasználási lehetőségeket, hiszen a hálózatok csak a tanításkor megismert objektumokat ismerik fel.

A tanítás vizsgálata meghaladta jelent tanulmány kereteit, eredetileg sem vállalkoztunk rá, de a sokféle kipróbált hálózat azt mutatta, hogy egy komplett computer vision alkalmazás fejlesztési idejének jelentős részét kell erre a feladatra fordítani.

ÖSSZEFOGLALÁS

A neurális hálózatok gyakorlati alkalmazásakor számtalan feltételt, jellemzőt kell figyelembe venni ahhoz, hogy a megoldás ne csak funkcionális választ adjon a feladatra, hanem egyfajta optimális megoldás legyen. Optimális abban az értelemben, hogy úgy teljesíti a funkcionális és nem funkcionális követelményeket, hogy egyben takarékos is. Takarékos a számítási kapacitással, a felhasznált hardver elemekkel, az energiafogyasztással, a sávszélességgel, a memóriával, a tárterülettel. Ugyanakkor teljesítménye és pontossága kielégíti az alkalmazás által elvárt értékeket.

Természetesen egyszerre nem lehet minden elvárásnak megfelelni. Különösen úgy, hogy ezek gyakran „egymás ellen dolgoznak". A tanulmányban láthattunk példát arra, hogy egy hálózat kisebb és gyorsabban futtattható változata számottevően pontatlanabb, mint a nagyobb, de egyben erőforrásigényesebb párja. Azt is láthattuk, hogy az erősebb számítási erőforrások (lásd erősebb Intel CPU-k) természetesen jobb teljesítménnyel futnak, de mindez visszaköszön a magasabb árukban is.

Ezek az optimalizálási feladatok gyakran megoldhatatlanok, ami alatt azt értjük, hogy a rendelkezésre álló erőforrásokkal nem lehet az alkalmazás kívánt céljait teljesíteni.

Például több és gyorsabb gépi látást szeretnénk egy konkrét hardveres környezetbe tenni, de azok teljesítménye nem elegendő a feladatra. Ezekre gyakran az a megoldás, hogy a számításigényes feladatot „kiszervezik" egy számítási felhőbe. Különösen a kisebb teljesítményű, végponti eszközökben fordul elő, hogy a feladathoz szükséges neurális hálózatot nem lokálisan futtatják.

Az utóbbi években megjelentek olyan eszközök, melyek a neurális hálók speciális számítási igényeit igyekeznek támogatni kisteljesítményű, végponti eszközökben. Ide sorolható az Intel Movidius VPU családja, melyet a saját eszköze (Intel Neural Compute Stick) mellett, más partnerek[13] is termékeikbe építettek. Tanulmányunkban azt vizsgáltuk, hogy mennyire könnyű ezzel az USB-s eszközzel dolgozni, mennyiben segíti az egyre több helyen megjelenő objektum detektálás feladatot. A mérések eredményeképpen az igazolódott, hogy nemcsak összevethető teljesítményű az idősebb, de még használatban lévő általános célú processzorokkal (Intel i5 CPU: 2nd / 4th generation), hanem önállóan is megállja a helyét az információs rendszerek végpontjain futó kis teljesítményű eszközökben.

Ezeknek a számítást támogató eszközöknek a fejlődése folyamatos. Az Intel mellett más vállalatok, például a Google és Nvidia is megjelentek a saját megoldásaikkal, ami azt vetíti előre, hogy a végponti (például IP kamera) eszközökben is egyre inkább megjelennek komolyabb gép látást igénylő alkalmazások. Ennek információbiztonsági vonatkozása is van. Egyrészt ezek az eszközök olyan komplex információkat fognak előállítani, továbbítani és adott esetben tárolni is, ami eddig csak a rendszerek adatközpontjában volt meg.

Gondoljunk arra, hogy egy gépi látást használó IP kamera is tudhatja majd, hogy pontosan kik és mikor közlekedtek a látómezejében. Az objektumazonosítás, az objektumkövetés segítségével pontos képe lehet a mozgó járművekről, tárgyakról. Ezek új információbiztonsági kihívást jelentenek az eszközök használóinak, de egyben új eszközök is lehetnek az információbiztonság fizikai védelmének biztosításához. Például egy okos kamera, mely nemcsak rögzíti az eseményeket, hanem elemzi is azt, képes arra, hogy ne csak általános mozgás érzékelése esetén küldjön riasztást, hanem összetettebb feltételeket is ki tudjon értékelni. Például riasztás munkaidőn kívüli teheráru forgalom észlelésekor vagy riasztás, ha egy rendezvényre többen léptek be, mint az előre meghatározott érték.

A gépi látást használó rendszerek alkalmazási lehetősége nagyon széles és különösen akkor lesz széles, ha általánosan elterjednek, mert lehetőség nyílik rá, hogy egyszerűbb eszközök is használhassák ezt a technológiát. Az Intel ezt a lehetőséget teremti meg a saját VPU (Visual Processing Unit) megoldásával.

[13] Intel VPU-t használó termékek: https://software.intel.com/content/www/us/en/develop/topics/iot/hardware/vision-accelerator-movidius-vpu.html

FELHASZNÁLT FORRÁSOK

[1] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo és A. Ylä-Jääski, *„Latency and Throughput Characterization of Convolutional Neural Networks for Mobile Computer Vision”* 2018. [Online]. https://arxiv.org/pdf/1803.09492.pdf. [hozzáférés dátuma: 2021. május 18.].

[2] Buzáné Kis P., *„Ismerkedés a TensorFlow rendszerrel”* [Online]. http://biointelligence.hu/pdf/tf_bkp.pdf. [hozzáférés dátuma: 2021. május 18.].

[3] O. Kharkovyna, *„Top 10 Best Deep Learning Frameworks in 2019,”* 2019. [Online]. https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de. [hozzáférés dátuma: 2021. május 18.].

[4] ONNX, *„ONNX Weboldal,”* [Online] https://onnx.ai/about.html. [hozzáférés dátuma: 2021. május 18.].

[5] Keras, 2020. [Online]. https://keras.io/. [hozzáférés dátuma: 2021. május 18.].

[6] B. Vision, *„Caffe,”* 2020. [Online]. https://caffe.berkeleyvision.org/. [hozzáférés dátuma: 2021. május 18.].

[7] https://pjreddie.com/darknet/ [Online] [hozzáférés dátuma: 2021. május 18.].

[8] J. Redmon, S. Divvala, R. Girshick és A. Farhadi, *„You Only Look Once: Unified, Real-Time Object Detection,”* 2016. [Online]. https://arxiv.org/pdf/1506.02640.pdf. [hozzáférés dátuma: 2021. május 18.].

[9] J. Redmon, *„Installing Darknet,”* [Online]. https://pjreddie.com/darknet/install/. [hozzáférés dátuma: 2021. május 18.].

[10] „Converting YOLO Models to the Intermediate Representation (IR),” Intel Corporation, [Online]. https://docs.openvinotoolkit.org/2020.1/_docs_MO_DG_prepare_model_convert_model_tf_specific_Convert_YOLO_From_Tensorflow.html. [hozzáférés dátuma: 2021. május 18.].